SIGNALS AND DAEMON PROCESSES

Divya Jennifer Dsouza
NMAMIT,Nitte

Signals are software interrupts. Signals provide a way of handling asynchronous events: a user at a terminal typing the interrupt key to stop a program or the next program in a pipeline terminating prematurely.

Name	Description	Default action
SIGABRT	abnormal termination (abort)	terminate+core
SIGALRM	timer expired (alarm)	terminate
SIGBUS	hardware fault	terminate+core
SIGCANCEL	threads library internal use	ignore
SIGCHLD	change in status of child	ignore
SIGCONT	continue stopped process	continue/ignore
SIGEMT	hardware fault	terminate+core
SIGFPE	arithmetic exception	terminate+core
SIGFREEZE	checkpoint freeze	ignore
SIGHUP	hangup	terminate
SIGILL	illegal instruction	terminate+core
SIGINFO	status request from keyboard	ignore
SIGINT	terminal interrupt character	terminate
SIGIO	asynchronous I/O	terminate/ignore
SIGIOT	hardware fault	terminate+core
SIGKILL	termination	terminate
SIGLWP	threads library internal use	ignore
SIGPIPE	write to pipe with no readers	terminate
SIGPOLL	pollable event (poll)	terminate
SIGPROF	profiling time alarm (setitimer)	terminate

SIGPWR	power fail/restart	terminate/ignore
SIGQUIT	terminal quit character	terminate+core
SIGSEGV	invalid memory reference	terminate+core
SIGSTKFLT	coprocessor stack fault	terminate
SIGSTOP	stop	stop process
SIGSYS	invalid system call	terminate+core
SIGTERM	termination	terminate
SIGTHAW	checkpoint thaw	ignore
SIGTRAP	hardware fault	terminate+core
SIGTSTP	terminal stop character	stop process
SIGTTIN	background read from control tty	stop process
SIGTTOU	background write to control tty	stop process
SIGURG	urgent condition (sockets)	ignore
SIGUSR1	user-defined signal	terminate
SIGUSR2	user-defined signal	terminate
SIGVTALRM	virtual time alarm (`setitimer`)	terminate
SIGWAITING	threads library internal use	ignore
SIGWINCH	terminal window size change	ignore
SIGXCPU	CPU limit exceeded (`setrlimit`)	terminate+core/ignore
SIGXFSZ	file size limit exceeded (`setrlimit`)	terminate+core/ignore
SIGXRES	resource control exceeded	Ignore

When a signal is sent to a process, it is pending on the process to handle it. The process can react to pending signals in one of three ways:

- Accept the **default action** of the signal, which for most signals will terminate the process.
- **Ignore the signal.** The signal will be discarded and it has no affect whatsoever on the recipient process.
- Invoke a **user-defined function.** The function is known as a signal handler routine and the signal is said to be *caught when this function is called.*

THE UNIX KERNEL SUPPORT OF SIGNALS

- When a signal is generated for a process, the kernel will set the corresponding signal flag in the process table slot of the recipient process.
- If the recipient process is asleep, the kernel will awaken the process by scheduling it.
- When the recipient process runs, the kernel will check the process U-area that contains an array of signal handling specifications.
- If array entry contains a zero value, the process will accept the default action of the signal.
- If array entry contains a 1 value, the process will ignore the signal and kernel will discard it.
- If array entry contains any other value, it is used as the function pointer for a user-defined signal handler routine.

Signals

- The function prototype of the signal API is:

#include <signal.h>

void (*signal(int sig_no, void (*handler)(int)))(int);

- The formal argument of the API are: sig_no is a signal identifier like SIGINT or SIGTERM.
- The handler argument is the function pointer of a user-defined signal handler function.

- The following example attempts to catch the SIGTERM signal, ignores the SIGINT signal, and accepts the default action of the SIGSEGV signal.

- The pause API suspends the calling process until it is interrupted by a signal and the corresponding signal handler does a return:

```
#include<iostream.h>
#include<signal.h>
/*signal handler function*/
void catch_sig(int sig_num)
{
signal (sig_num,catch_sig);


cout<<"catch_sig:"<<sig_num<<endl;
}

/*main function*/
int main()
{
}
signal(SIGTERM,catch_sig);
signal(SIGINT,SIG_IGN);
signal(SIGSEGV,SIG_DFL);
pause( );    /*wait for a signal interruption*/
```

- The SIG_IGN specifies a signal is to be ignored, which means that if the signal is generated to the process, it will be discarded without any interruption of the process.

- The SIG_DFL specifies to accept the default action of a signal.